View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Procedia Computer Science

Volume 80, 2016, Pages 86–97

ICCS 2016. The International Conference on Computational Science

High Performance LDA through Collective Model Communication Optimization

Bingjing Zhang[1], Bo Peng[1,2], and Judy Qiu[1]

[1] Indiana University, Bloomington, Indiana, U.S.A.
{zhangbj,pengb,xqiu}@indiana.edu
[2] Peking University, Beijing, China

Abstract

LDA is a widely used machine learning technique for big data analysis. The application includes an inference algorithm that iteratively updates a model until it converges. A major challenge is the scaling issue in parallelization owing to the fact that the model size is huge and parallel workers need to communicate the model continually. We identify three important features of the model in parallel LDA computation: 1. The volume of model parameters required for local computation is high; 2. The time complexity of local computation is proportional to the required model size; 3. The model size shrinks as it converges. By investigating collective and asynchronous methods for model communication in different tools, we discover that optimized collective communication can improve the model update speed, thus allowing the model to converge faster. The performance improvement derives not only from accelerated communication but also from reduced iteration computation time as the model size shrinks during the model convergence. To foster faster model convergence, we design new collective communication abstractions and implement two Harp-LDA applications, "lgs" and "rtt". We compare our new approach with Yahoo! LDA and Petuum LDA, two leading implementations favoring asynchronous communication methods in the field, on a 100-node, 4000-thread Intel Haswell cluster. The experiments show that "lgs" can reach higher model likelihood with shorter or similar execution time compared with Yahoo! LDA, while "rtt" can run up to 3.9 times faster compared with Petuum LDA when achieving similar model likelihood.

Keywords: Latent Dirichlet Allocation, Parallel Algorithm, Big Model, Communication Model, Communication Optimization

1 Introduction

Latent Dirichlet Allocation (LDA) [1] is an important machine learning technique that has been widely used in areas such as text mining, advertising, recommender systems, network analysis, and genetics. Though extensive research on this topic exists, the data analysis community is

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

© The Authors. Published by Elsevier B.V.

doi:10.1016/j.procs.2016.05.300

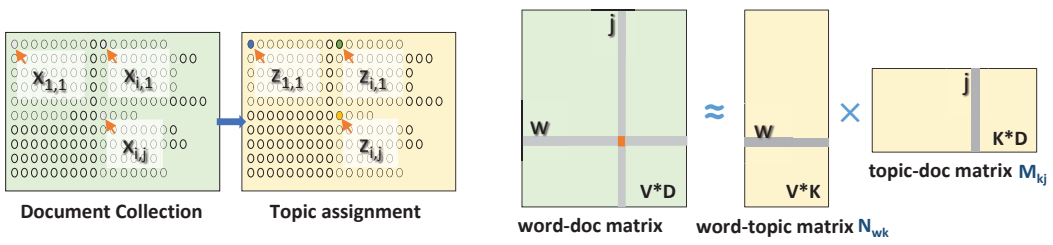


Figure 1: (a) Topics Discovery (b) View of Matrix Decomposition

still endeavoring to scale it to web-scale corpora to explore more subtle semantics with a big model which may contain billions of model parameters [5].

We identify that the size of the model required for local computation is so large that sending such data to every worker results in communication bottlenecks. The required computation time is great due to the large model size. In addition, the model size shrinks as the model converges. As a result, a faster communication method can speed up the model convergence, in which the model size shrinks and reduces the iteration execution time.

By guaranteeing the algorithm correctness, various model communication strategies are developed in parallel LDA. Existing solutions favor asynchronous communication methods, since it not only avoids global waiting but also quickly makes the model update visible to other workers and thereby boosts model convergence. We propose a more efficient approach in which the model communication speed is improved upon with optimized collective communication methods. We abstract three new communication operations and implement them on top of Harp [15]. We develop two Harp-LDA applications and compare them with Yahoo! LDA[1] and Petuum LDA[2], two well-known implementations in the domain. This is done on three datasets each with a total of 10 billion model parameters. The results on a 100-node, 4000-thread Intel Haswell cluster show that collective communication optimizations can significantly reduce communication overhead and improve model convergence speed.

The following sections describe: the background of LDA application (Section 2), the big model problem in parallel LDA (Section 3), the model communication challenges in parallel LDA and related work (Section 4), Harp-LDA implementations (Section 5), experiments and performance comparisons (Section 6), and the conclusion (Section 7).

2 Background

LDA modeling techniques can find latent structures inside the training data which are abstracted as a collection of documents, each with a bag of words. It models each document as a mixture of latent topics and each topic as a multinomial distribution over words. Then an inference algorithm works iteratively until it outputs the converged topic assignments for the training data (see Figure 1(a)). Similar to Singular Value Decomposition (see Figure 1(b)), LDA can be viewed as a sparse matrix decomposition technique on a word-document matrix, but it roots on a probabilistic foundation and has different computation characteristics.

[1] https://github.com/sudar/Yahoo_LDA
[2] https://github.com/petuum/bosen/wiki/Latent-Dirichlet-Allocation

Among the inference algorithms for LDA, Collapsed Gibbs Sampling (CGS) [12] shows high scalability in parallelization [3, 11], especially compared with another commonly used algorithm, Collapsed Variational Bayes (CVB[3]) [1]. CGS is a Markov chain Monte Carlo type algorithm. In the "initialize" phase, each training data point, or token, is assigned to a random topic denoted as z_{ij}. Then it begins to reassign topics to each token $x_{ij} = w$ by sampling from a multinomial distribution of a conditional probability of z_{ij}:

$$p\left(z_{ij} = k \mid z^{\neg ij}, x, \alpha, \beta\right) \propto \frac{N_{wk}^{\neg ij} + \beta}{\sum_w N_{wk}^{\neg ij} + V\beta} \left(M_{kj}^{\neg ij} + \alpha\right) \tag{1}$$

Here superscript $\neg ij$ means that the corresponding token is excluded. V is the vocabulary size. N_{wk} is the token count of word w assigned to topic k in K topics, and M_{kj} is the token count of topic k assigned in document j. The matrices Z_{ij}, N_{wk} and M_{kj}, are the model. Hyperparameters α and β control the topic density in the final model output. The model gradually converges during the process of iterative sampling. This is the phase where the "burn-in" stage occurs and finally reaches the "stationary" stage.

The sampling performance is more memory-bound than CPU-bound. The computation itself is simple, but it relies on accessing two large sparse model matrices in the memory. In Algorithm 1, sampling occurs by the column order of the word-document matrix, called

Algorithm 1: LDA Collapsed Gibbs Sampling Algorithm

input : training data X, the number of topics K, hyperparamters α, β
output: topic assignment matrix Z_{ij}, topic-document matrix M_{kj}, word-topic matrix N_{wk}

1 Initialize M_{kj}, N_{wk} to zeros `// Initialize phase`
2 **foreach** *document* $j \in [1, D]$ **do**
3 **foreach** *token position* i *in document* j **do**
4 $z_{i,j} = k \sim Mult(\frac{1}{K})$ `// sample topics by multinomial distribution`
5 $m_{k,j}$ += 1; $n_{w,k}$ += 1 `// token` $x_{i,j}$ `is word` w`, update the model matrices`
  `// Burn-in and Stationary phase`
6 **repeat**
7 **foreach** *document* $j \in [1, D]$ **do**
8 **foreach** *token position* i *in document* j **do**
9 $m_{k,j}$ −= 1; $n_{w,k}$ −= 1 `// decrease counts`
10 $z_{i,j} = k' \sim p(z_{i,j} = k|rest)$ `// sample a new topic by Equation (1)`
11 $m_{k',j}$ += 1; $n_{w,k'}$ += 1 `// increase counts for the new topic`
12 **until** *convergence*;

"sample by document". Although M_{kj} is cached when sampling all the tokens in a document j, the memory access to N_{wk} is random since tokens are from different words. Symmetrically, sampling can occur by the row order, called "sample by word". In both cases, the computation time complexity is highly related to the size of model matrices. SparseLDA [14] is an optimized CGS sampling implementation mostly used in state-of-the-art LDA trainers. In Line 10 of Algorithm 1, the conditional probability is usually computed for each k with a total of K times, but SparseLDA decreases the time complexity to the number of non-zero items in one row of N_{wk} and in one column of M_{kj}, both of which are much smaller than K on average.

[3]CVB algorithm is used in Spark LDA (http://spark.apache.org/docs/latest/mllib-clustering.html) and Mahout LDA (https://mahout.apache.org/users/clustering/latent-dirichlet-allocation.html)

3 Big Model Problem in Parallel LDA

Sampling on Z_{ij} in CGS is a strict sequential procedure, although it can be parallelized without much loss in accuracy [3]. Parallel LDA can be performed in a distributed environment or a shared-memory environment. Due to the huge volume of training documents, we focus on the distributed environment which is formed by a number of compute nodes deployed with a single worker process apiece. Every worker takes a partition of the training document set and performs the sampling procedure with multiple threads. The workers either communicate through point-to-point communication or collective communication.

The LDA model contains four parts: I. Z_{ij} - topic assignments on tokens, II. N_{wk} - token counts of words on topics (word-topic matrix), III. M_{kj} - token counts of documents on topics (topic-document matrix), and IV. $\sum_w N_{wk}$ - token counts of topics. Here Z_{ij} is always stored along with the training tokens. For the other three, because the training tokens are partitioned by document, M_{kj} is stored locally while N_{wk} and $\sum_w N_{wk}$ are shared. For the shared model parts, a parallel LDA implementation may use the latest model or the stale model in the sampling procedure. The latest model means the current model parameters used in computation are up-to-date and not modified simultaneously by other workers, while the stale model means the model parameters are old. We show that model consistency is important to convergence speed in Section 6.

Now we calculate the size of N_{wk}, a huge but sparse $V * K$ matrix. We note that the word distribution in the training data generally follows a power law. If we sort the words based on their frequencies from high to low, for a word with rank i, its frequency is $freq(i) = C * i^{-\lambda}$. Then for W, the total number of training tokens, we have

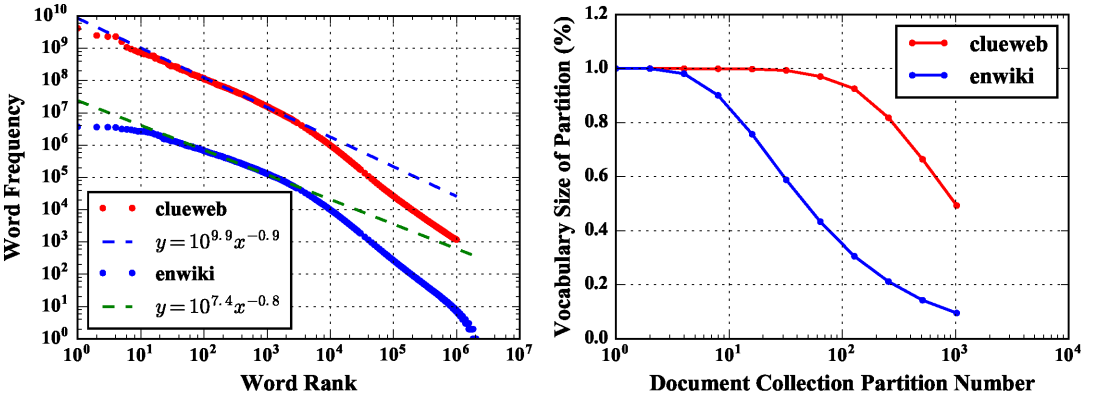
$$W = \sum_{i=1}^{V} freq(i) = \sum_{i=1}^{V} (C * i^{-\lambda}) \approx C * (lnV + \gamma + \frac{1}{2V}) \tag{2}$$

Here λ is a constant near 1, constant $C = freq(1)$. To simplify the analysis, we consider $\lambda = 1$. Then W is the partial sum of the harmonic series which have logarithmic growth, where γ is the Euler-Mascheroni constant ≈ 0.57721. The real model size (denoted as S) depends on the count of non-zero cells. In the "initialize" phase of CGS, with random topic assignment, a word i gets $max(K, freq(i))$ non-zero cells. If $freq(J) = K$, then $J = C/K$, and we get:

$$S_{init} = \sum_{i=1}^{J} K + \sum_{i=J+1}^{V} freq(i) = W - \sum_{i=1}^{J} freq(i) + \sum_{i=1}^{J} K = C * (lnV + lnK - lnC + 1) \tag{3}$$

The actual initial model size S_{init} is logarithmic to matrix size $V * K$, which means $S << V * K$. However this does not mean S_{init} is small. Since C can be very large, even $C * ln(V * K)$ can result in a large number. In the progress of iterations, the model size shrinks as the model converges. When a stationary state is reached, the average number of topics per word drops to a certain small constant ratio of K, which is determined by the concentration parameters α, β and the nature of the training data itself.

The local vocabulary size V' of each worker determines the model volume required for computation. When documents are randomly partitioned to N processes, every word with a frequency larger than N has a high probability of occurring on all the processes. If $freq(L) = N$ at rank L, we get: $L = \frac{W}{(lnV+\gamma)*N}$. For a large training dataset, the ratio between L and V is often very high, indicating that local computation requires most of the model parameters. Figure 2 shows the difficulty of controlling local vocabulary size in random document partitioning. When 10 times more partitions are introduced, there is only a sub-linear decrease in the



(a) (b)

Figure 2: (a) Zipf's Law of the Word Frequencies (b) Vocabulary Size vs. Document Partitioning

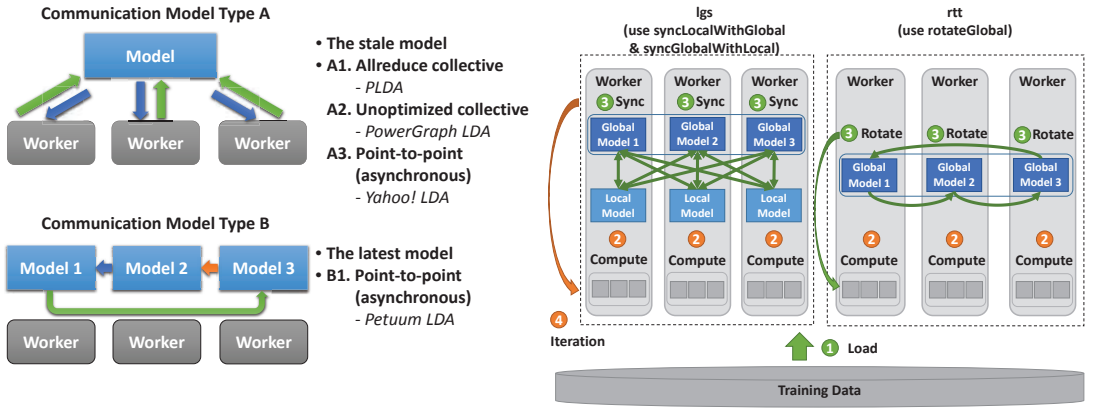
vocabulary size per partition. We will use "clueweb" and "enwiki" datasets as examples (the contents of these datasets are discussed in Section 6). In "clueweb", each partition gets 92.5% of V when the training documents are randomly split into 128 partitions. "enwiki" is around 12 times smaller than "clueweb". It gets 90% of V with 8 partitions, keeping a similar ratio. In summary, though the local model size reduces as the number of compute nodes grows, it is still a high percentage of V in many situations.

4 Model Communication Challenges in Parallel LDA and Related Work

The analysis in previous sections shows three key properties of the big LDA model: 1. The initial model size is huge but it reduces as the model converges; 2. The model parameters required in local computation is a high percentage of all the model parameters; 3. The local computation time is related to the local model size. These properties indicate that model communication optimization is necessary because it can accelerate the model update process and result in a huge benefit in computation and communication of later iterations. Of the various communication methods used in state-of-the-art implementations, we abstract them into two types of communication models (see Figure 3(a)).

In Communication Model Type A, the computation occurs on the stale model. Before performing the sampling procedure, workers fetch the related model parameters to the local memory. After computation, they send updates back to the model. There are many communication models in this category. In A1, without storing a shared model, it synchronizes local model parameters through an "allreduce" operation [4]. PLDA [10] follows this communication model. "allreduce" is routing optimized, but it does not consider the model requirement in local computation, causing high memory usage and high communication load. In A2, model parameters are fetched and returned directly in a collective way. PowerGraph LDA[4] follows this communication model [6]. Though it communicates less model parameters compared with A1, the performance is low for lack of routing optimization. A more popular communication model

[4] https://github.com/dato-code/PowerGraph/tree/master/toolkits/topic_modeling



(a) (b)

Figure 3: (a) Communication Models (b) Harp-LDA Implementations

is A3, which uses asynchronous point-to-point communication. Yahoo! LDA [13, 2] and Parameter Server [7] follow this communication model. In A3, each worker independently fetches and updates the related model parameters without waiting for other workers. A3 can ease the communication overhead, however, its model update rate is not guaranteed. A word's model parameters may be updated either by changes from all the training tokens, a part of them, or even no change. A solution to this problem is to combine A3 and A2. This is implemented in Petuum (version 0.93) LDA [8].

In Communication Model Type B, each worker first takes a partition of the model parameters, after which the model partitions are "shifted" between workers. When all the partitions are accessed by all the workers, an iteration is completed. There is only one communication model B1 which uses asynchronous point-to-point communication. Petuum (version 1.1) LDA [9] follows this model.

A better solution for Communication Model Type A can be a conjunction of A1 and A2 with new collective communication abstractions. There are three advantages to such a strategy. First, considering the model requirement in local computation, it reduces the model parameters communicated. Second, it optimizes routing through searching "one-to-all" communication patterns. Finally, it maintains the model update rate compared with asynchronous methods. For Communication Model Type B, using collective communication is also helpful because it maximizes bandwidth usage between compute nodes and avoids flooding the network with small messages, which is what B1 does.

5 Harp-LDA Implementations

Based on the analysis above, we parallelize LDA with optimized collective communication abstractions on top of Harp [15], a collective communication library plugged into Hadoop[5]. We use "table" abstractions defined in Harp to organize the shared model parameters. Each table may contain one or more model partitions, and the tables defined on different processes are associated to manage a distributed model. We partition the model parameters based on the

[5] http://hadoop.apache.org

range of word frequencies in the training dataset. The lower the frequency of the word, the higher the partition ID given. Then we map partition IDs to process IDs based on the modulo operation. In this way, each process contains model partitions with words whose frequencies are ranked from high to low.

We add three collective communication operations. The first two operations, "syncGlobalWithLocal" and "syncLocalWithGlobal", are paired. Here another type of table is defined to describe the local models. Each partition in these tables is considered a local version of a global partition according to the corresponding ID. "syncGlobalWithLocal" merges partitions from different local model tables to one in the global tables while "syncLocalWithGlobal" redistributes the model partitions in the global tables to local tables. Routing optimized broadcasting [4] is used if "one-to-all" communication patterns are detected. Lastly, "rotateGlobal" considers processes in a ring topology and shifts the model partitions from one process to the next neighbor.

We present two parallel LDA implementations. One is "lgs", which uses "syncGlobalWithLocal" paired with "syncLocalWithGlobal". Another is "rtt", which uses "rotateGlobal" (see Figure 3(b)). In both implementations, the computation and communication are pipelined, i.e., the model parameters are sliced in two and communicated in turns. Model Part IV is synchronized through A1 at the end of every iteration. The SparseLDA algorithm is used for the sampling procedure. "lgs" samples by document while "rtt" samples by word. This is done to keep the consistency between implementations for unbiased communication performance comparisons in future experiments.

6 Experiments

Experiments are done on a cluster[6] with Intel Haswell architecture. This cluster contains 32 nodes each with two 18-core 36-thread Xeon E5-2699 processors and 96 nodes each with two 12-core 24-thread Xeon E5-2670 processors. All the nodes have 128GB memory and are connected with 1Gbps Ethernet (eth) and Infiniband (ib). For testing, 31 nodes with Xeon E5-2699 and 69 nodes with Xeon E5-2670 are used to form a cluster of 100 nodes, each with 40 threads. All the tests are done with Infiniband through IPoIB support.

"clueweb"[7], "enwiki", and "bi-gram"[8] are three datasets used (see Table 1). The model parameter settings are comparable to other research work [5], each with a total of 10 billion parameters. α and β are both fixed to a commonly used value 0.01 to exclude dynamic tuning. We test several implementations: "lgs", "lgs-4s" ("lgs" with 4 rounds of model synchronization per iteration, each round with 1/4 of the training tokens), and "rtt". To evaluate the quality of the model outputs, we use the model likelihood on the training dataset to monitor model convergence. We compare our implementations with two leading implementations, Yahoo! LDA and Petuum LDA, and thereby learn how communication methods affect LDA performance by studying the model convergence speed.

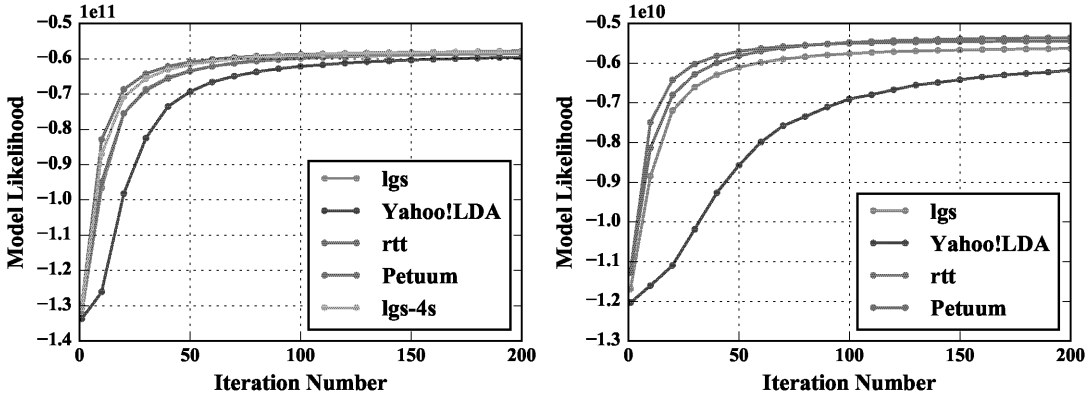
6.1 Model Convergence Speed Measured by Iteration

We compare the model convergence speed by analyzing model outputs on Iteration 1, 10, 20... 200. In an iteration, every training token is sampled once. Thus the number of model updates

[6] https://portal.futuresystems.org
[7] 10% of ClueWeb09 (a collection of English web pages, http://lemurproject.org/clueweb09.php/)
[8] Both "enwiki" and "bi-gram" are English articles from Wikipedia (https://www.wikipedia.org)

Dataset	Number of Docs	Number of Tokens	Vocabulary	Doc Length Mean/SD	Number of Topics	Initial Model Size
clueweb	50.5M	12.4B	1M	224/352	10K	14.7GB
enwiki	3.8M	1.1B	1M	293/523	10K	2.0GB
bi-gram	3.9M	1.7B	20M	434/776	500	5.9GB

Table 1: Training Data Settings in the Experiments



(a) (b)

Figure 4: (a) Model Convergence Speed on "clueweb" (b) Model Convergence Speed on "enwiki"

in each iteration is equal. Then we see how the model converges with the same amount of model updates.

On "clueweb" (see Figure 4(a)), Petuum has the highest model likelihood on all iterations. Due to "rtt"'s preference of using stale thread-local model parameters in multi-thread sampling, the convergence speed is slower. The lines of "rtt" and "lgs" are overlapped for their similar convergence speeds. In contrast to "lgs", the convergence speed of "lgs-4s" is as high as Petuum. This shows that increasing the number of model update rounds improves convergence speed. Yahoo! LDA has the slowest convergence speed because asynchronous communication does not guarantee all model updates are seen in each iteration. On "enwiki" (see Figure 4(b)), as before, Petuum achieves the highest accuracy out of all iterations. "rtt" converges to the same model likelihood level as Petuum at iteration 200. "lgs" demonstrates slower convergence speed but still achieves high model likelihood, while Yahoo! LDA has both the slowest convergence speed and the lowest model likelihood at iteration 200.

These results match our previous analysis. Though the number of model updates is the same, an implementation using the stale model converges slower than one using the latest model. For those using the stale model, "lgs-4s" is faster than "lgs" while "lgs" is faster than Yahoo! LDA. This means by increasing the number of model update rounds, the model parameters used in computation are newer, and the convergence speed is improved.

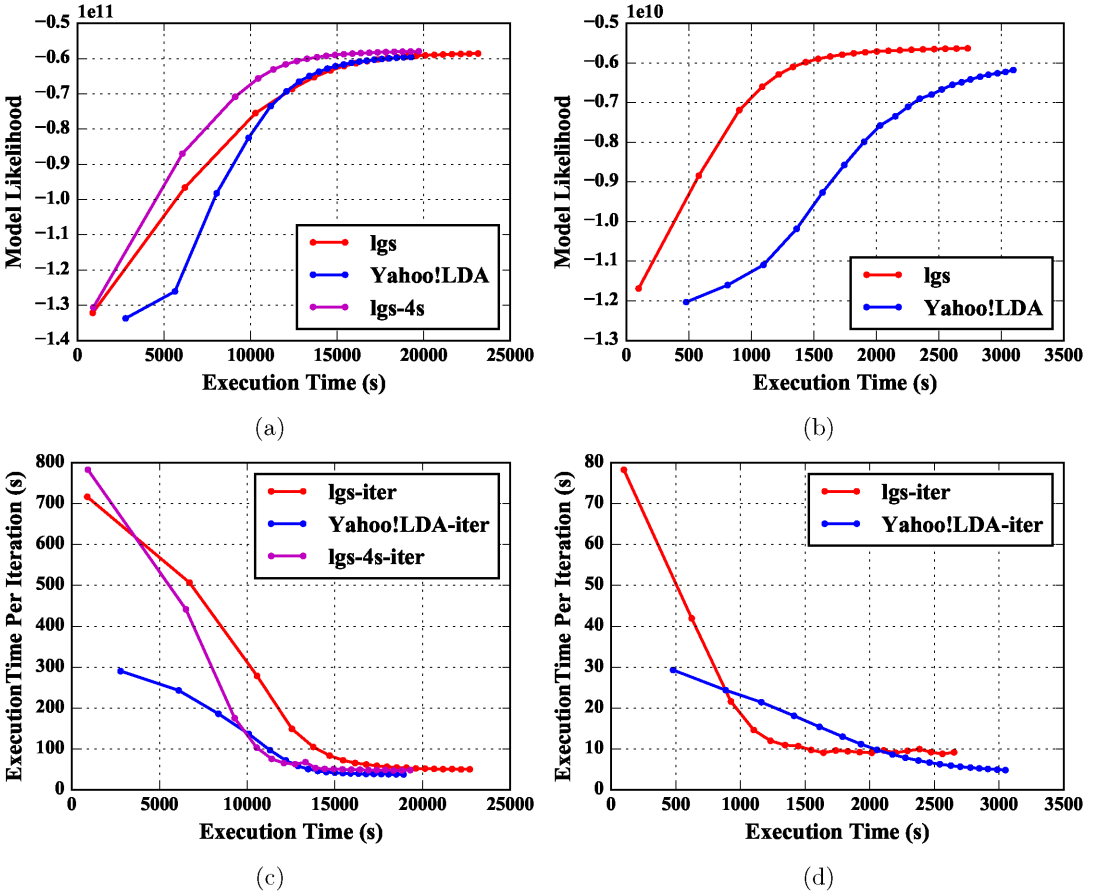


Figure 5: Model Likelihood vs. Elapsed Time on (a) "clueweb" (b) "enwiki"; Iteration Time vs. Elapsed Time on (c) "clueweb" (d) "enwiki"

6.2 Model Convergence Speed Measured by Elapsed Time

We first compare the execution speed between "lgs" and Yahoo! LDA. On "clueweb", we show the convergence speed based on the elapsed execution time (see Figure 5(a)). Yahoo! LDA takes more time to finish Iteration 1 due to its slow model initialization, which demonstrates that it has a sizable overhead on the communication end. In later iterations, while "lgs" converges faster, Yahoo! LDA catches up after 30 iterations. This observation can be explained by our slower computation speed. To counteract the computation overhead, we increase the number of model synchronization rounds per iteration to four. Thus the computation overhead is reduced by using a newer and smaller model. Although the execution time for "lgs-4s" is still slightly longer than Yahoo! LDA, it obtains higher model likelihood and maintains faster convergence speed during the whole execution.

Similar results are shown on "enwiki", but this time "lgs" not only achieves higher model likelihood but also has faster model convergence speed throughout the whole execution (see Figure 5(b)). From both experiments, we learn that though the computation is slow in "lgs",

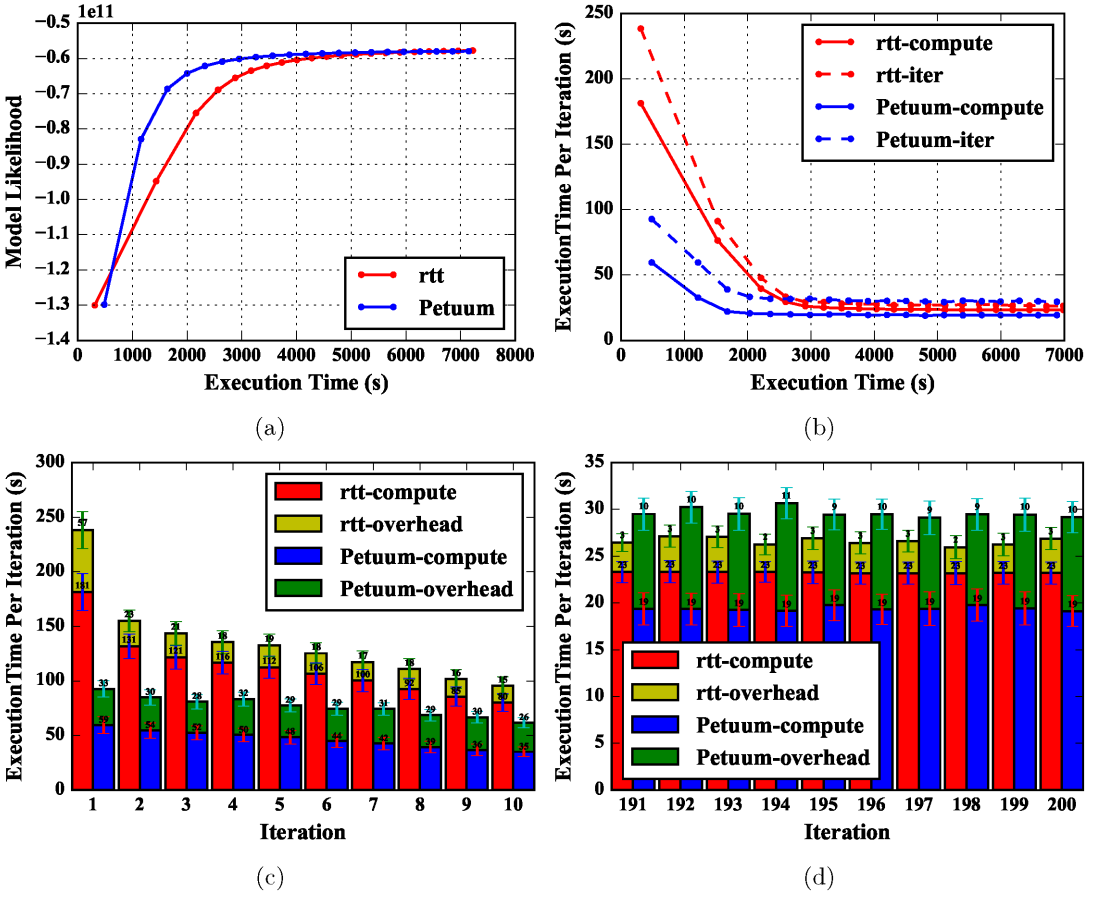


Figure 6: Comparison on "clueweb" (a) Model Likelihood vs. Elapsed Time (b) Iteration Time vs. Elapsed Time' (c) First 10 Iteration Times (d) Final 10 Iteration Times

with collective communication optimization, the model size quickly shrinks so that its computation time is reduced significantly. At the same time, although Yahoo! LDA does not have any extra overhead other than computation in each iteration, its iteration execution time reduces slowly because it keeps computing with an older model (see Figure 5(c)(d)).

Next we compare "rtt" and Petuum LDA on "clueweb"and "bi-gram". On "clueweb", the execution times and model likelihood achieved on both sides are similar (see Figure 6(a)). Both are around 2.7 times faster than the results in "lgs" and Yahoo! LDA. This is because they use the latest model parameters for sampling, and using the "sample by word" method leads to better performance. Though "rtt" has higher computation time compared with Petuum LDA, the communication overhead per iteration is lower. When the execution arrives at the final few iterations, while computation time per iteration in "rtt" is higher, the whole execution time per iteration becomes lower (see Figure 6(b)(c)(d)). This is because Petuum communicates each word's model parameters in small messages and generates high overhead. On "bi-gram", the results show that Petuum does not perform well when the number of words in the model increases. The high overhead in communication causes the convergence speed to be slow,

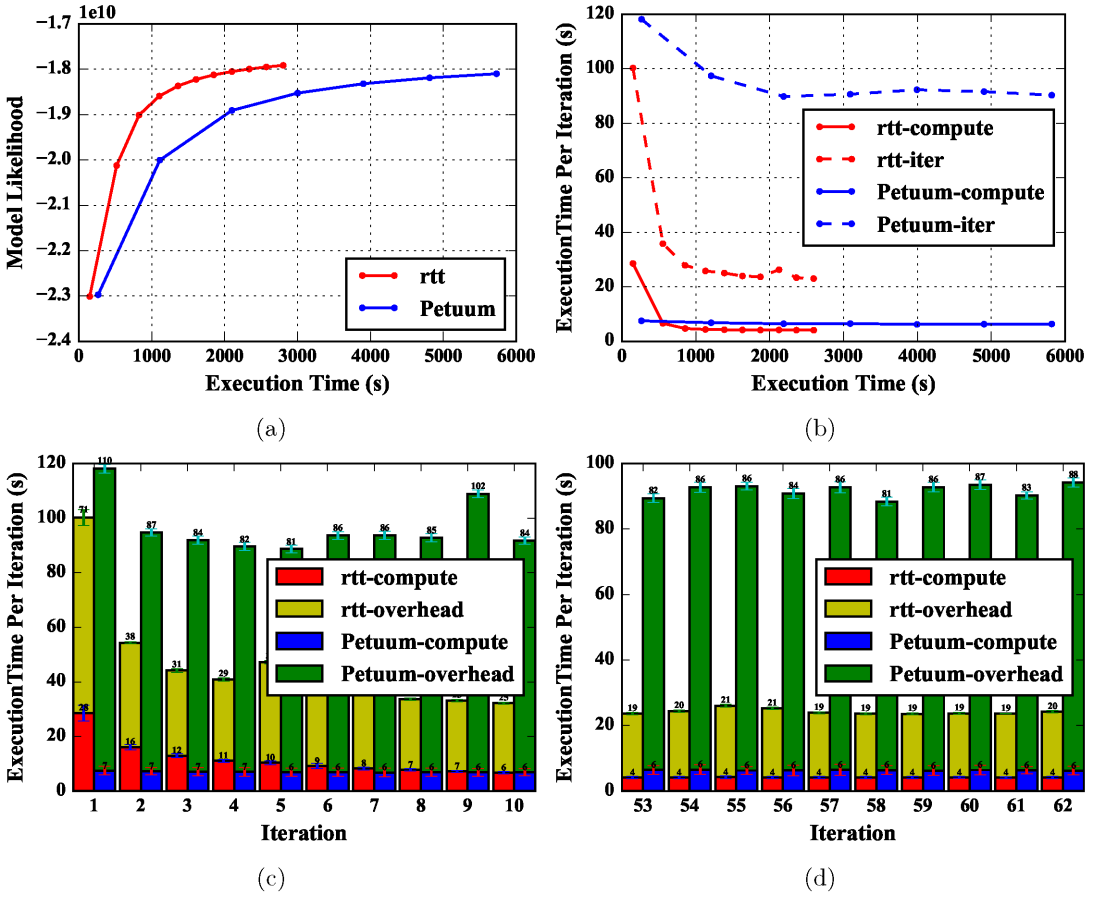


Figure 7: Comparison on "bi-gram" (a) Model Likelihood vs. Elapsed Time (b) Iteration Time vs. Elapsed Time (c) First 10 Iteration Times (d) Final 10 Iteration Times

and Petuum cannot even continue executing after 60 iterations due to a memory outage (see Figure 7(a)). Figure 7(b)(c)(d) shows this performance difference on communication.

7 Conclusion

Through the analysis on the LDA model, we identify three model properties in parallel LDA computation: 1. The model requirement in local computation is high; 2. The time complexity of local sampling is related to the required model size; 3. The model size shrinks as it converges. These properties suggest that using collective communication optimizations can improve the model update speed, which allows the model to converge faster. When the model converges quickly, its size shrinks greatly, and the iteration execution time also reduces. We show that optimized collective communication methods perform better than asynchronous methods in parallel LDA. "lgs" results in faster model convergence and higher model likelihood at iteration 200 compared to Yahoo! LDA. On "bi-gram", "rtt" shows significantly lower communication

overhead than Petuum LDA, and the total execution time of "rtt" is 3.9 times faster. On "clueweb", although the computation speed of the first iteration is 2- to 3-fold slower, the total execution time remains similar.

Despite the implementation differences between "rtt", "lgs", Yahoo! LDA, and Petuum LDA, the advantages of collective communication methods are evident. Compared with asynchronous communication methods, collective communication methods can optimize routing between parallel workers and maximize bandwidth utilization. Though collective communication will result in global waiting, the resulting overhead is not as high as speculated. The chain reaction set off by improving the LDA model update speed amplifies the benefits of using collective communication methods. In future work, we will focus on improving intra-node LDA performance in many-core systems and apply our model communication strategies to other machine learning algorithms facing big model problems.

Acknowledgments

We gratefully acknowledge support from Intel Parallel Computing Center Grant, NSF 1443054 CIF21 DIBBs 1443054 Grant, and NSF OCI 1149432 CAREER Grant. We appreciate the system support offered by FutureSystems.

References

[1] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[2] A. Ahmed et al. Scalable Inference in Latent Variable Models. In *WSDM*, 2012.

[3] D. Newman et al. Distributed Algorithms for Topic Models. *The Journal of Machine Learning Research*, 10:1801–1828, 2009.

[4] E. Chan et al. Collective Communication: Theory, Practice, and Experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[5] E. Xing et al. Petuum: A New Platform for Distributed Machine Learning on Big Data. In *KDD*, 2015.

[6] J. Gonzalez et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *OSDI*, 2012.

[7] M. Li et al. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*, 2014.

[8] Q. Ho et al. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *NIPS*, 2013.

[9] S. Lee et al. On Model Parallelization and Scheduling Strategies for Distributed Machine Learning. In *NIPS*, 2014.

[10] Y. Wang et al. PLDA: Parallel Latent Dirichlet Allocation for Large-scale Applications. In *Algorithmic Aspects in Information and Management*, pages 301–314. Springer, 2009.

[11] Y. Wang et al. Peacock: Learning Long-Tail Topic Features for Industrial Applications. *ACM Transactions on Intelligent Systems and Technology*, 6(4), 2015.

[12] P. Resnik and E. Hardist. Gibbs Sampling for the Uninitiated. Technical report, University of Maryland, 2010.

[13] A. Smola and S. Narayanamurthy. An Architecture for Parallel Topic Models. *Proceedings of the VLDB Endowment*, 3(1-2):703–710, 2010.

[14] L. Yao, D. Mimno, and A. McCallum. Efficient Methods for Topic Model Inference on Streaming Document Collections. In *KDD*, 2009.

[15] B. Zhang, Y. Ruan, and J. Qiu. Harp: Collective Communication on Hadoop. In *IC2E*, 2014.